Systèmes de transitions - Modélisation TLA$^+$

Durée 1h30 - Documents autorisés

5 avril 2019

1 Questions de cours (2 points)

Soit deux variables x, un entier, et t un tableau inclus dans $[Nat \to Nat]$.

1. Donner un prédicat TLA$^+$ qui exprime que x est plus grand que toute valeur de t.
   $\forall i \in \text{DOMAIN } t : x > t[i]$
2. Donner une action TLA$^+$ qui incrémente la x-ième case du tableau t, à condition que x soit dans le domaine de définition de t.
   $Act \triangleq x \in \text{DOMAIN } t \land t' = [t \text{ EXCEPT } ![x] = t[x] + 1] \land \text{UNCHANGED } x$

2 Exercice (6 points)

Soit le module TLA$^+$ ci-dessous définissant le système de transitions `Spec`.

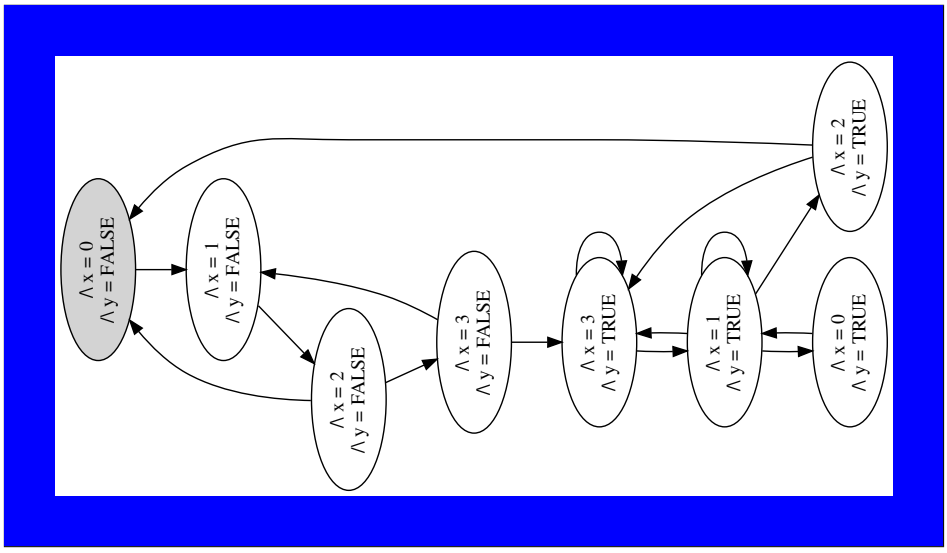
MODULE *examen18_test*

EXTENDS *Naturals*
VARIABLES x, y

$TypeInvariant \triangleq x \in Nat \land y \in \text{BOOLEAN}$

$DecX \triangleq x' = x - 2 \land x' \geq 0 \land y' = (y \land (x' \neq 0))$
$IncX \triangleq x < 3 \land x' = x + 1 \land \text{UNCHANGED } y$
$FreeX \triangleq y \land x = 1 \land x' \in \{0, 1, 2, 3\} \land \text{UNCHANGED } y$
$SetY \triangleq x = 3 \land y' = \text{TRUE} \land \text{UNCHANGED } x$
$Fairness \triangleq \text{WF}_{\langle x, y \rangle}(IncX) \land \text{SF}_{\langle x, y \rangle}(SetY)$
$Init \triangleq x = 0 \land y = \text{FALSE}$
$Spec \triangleq Init \land \Box[DecX \lor IncX \lor FreeX \lor SetY]_{\langle x, y \rangle} \land Fairness$

1. Dessiner le graphe d'exécution du système de transition.



2. Indiquer si les propriétés suivantes, exprimées en logique LTL ou CTL, sont vérifiées. Justifier les réponses (argumentaire ou contre-exemple).

   (a) $\Box\neg(x = 0 \wedge y)$
   faux : $x = 0 \wedge y$ accessible depuis l'état initial

   (b) $\Box\Diamond y$
   faux : $x = 0 \to 1 \to 2 \to 0$, avec y restant faux tjs

   (c) $\Box\Diamond(x \geq 2 \vee y)$
   vrai cf graphe : l'équité force à passer par $x = 2 \wedge \neg y$

   (d) $y \rightsquigarrow x = 1$
   vrai ?

   (e) $\exists\Diamond(x = 2 \wedge y)$
   vrai : état accessible depuis l'état initial

   (f) $\exists\Box(x \neq 3)$
   vrai : $(x = 0 \to x = 1 \to x = 2 \to x = 0)^\omega$ est une exécution valide.

   (g) $(x = 2)\ \exists\mathcal{U}\ (x = 1)$
   faux : $x = 2$ n'a pas de transition directe vers $x = 1$

   (h) $\forall\Box(y \Rightarrow \exists\Diamond(x = 1))$
   vrai : $x = 1$ tjs accessible depuis un état y.

3 Problème : validation à deux phases (12 points [1])

On considère le problème de la validation à deux phases avec un coordinateur. Un ensemble d'agents se coordonne pour prendre une décision. Si au moins l'un d'entre eux souhaite abandonner (*abort*), alors tous doivent décider d'abandonner ; si tous souhaitent valider (*commit*), alors tous doivent valider. Pour réaliser cela, chaque agent envoie sa proposition à un coordinateur. Si le coordinateur reçoit un abort, il diffuse à tous la décision d'abandon ; s'il reçoit autant de proposition de commit qu'il y a d'agents, il diffuse à tous la décision de valider.

Un squelette de module TLA$^+$ `TwoPhaseCommit.tla` est fourni à la fin du sujet.

3.1 Module complet

1. Compléter l'action `CoordDecideCommit` pour que le coordinateur décide `committed` et envoie à tous les agents cette décision, à condition que tous les agents aient proposé `commit`.

```
CoordDecideCommit ==
   ∧ cstate = "undecided"
   ∧ commits = N
   ∧ cstate' = "committed"
   ∧ amsgs' = [ i ∈ Agent |-> amsgs[i] ∪ {cstate'} ]
   ∧ UNCHANGED <<astate,cmsgs,commits>>
```

2. Définir le prédicat de transition `NextAgent(i)` qui représente les transitions possibles pour un agent i.
   $NextAgent(i) \triangleq AgentProposeAbort(i) \vee AgentProposeCommit(i) \vee AgentReceiveDecision(i)$

1. Toutes les questions valent autant sauf la 15 qui vaut double.

3. Définir le prédicat de transition `NextCoord` qui représente les transitions possibles pour le coordinateur.
   $NextCoord \triangleq CoordReceiveCommit \lor CoordReceiveAbort \lor CoordDecideCommit$
4. Définir le prédicat de transition `Next` qui représente toutes les transitions possibles du système.
   $Next \triangleq (\exists i \in Agent : NextAgent(i)) \lor NextCoord$
5. Définir la propriété `Spec` qui décrit le système de transitions.
   $Init \land \Box[Next]_{vars} \land Fairness$

3.2 Spécification

Exprimer en TLA$^+$ les propriétés suivantes (qui ne sont pas nécessairement vérifiées par le modèle) :

6. `FinalAgreement` : deux agents ne peuvent pas décider l'un `committed` et l'autre `aborted`.
   $\Box(\neg\exists ag1, ag2 \in Agent : astate[ag1] = "committed" \land astate[ag2] = "aborted")$
7. `CommitOnAgreement` : si le coordinateur décide `committed`, c'est que tous les agents sont prêts à valider ou ont validé.
   $\Box(cstate = "committed" \Rightarrow \forall ag \in Agent : astate[ag] \in \{"readyCommit", "committed"\})$
8. `AbortOnVeto` : le coordinateur décide `aborted` si au moins un agent abandonne.
   $\Box(cstate = "aborted" \Rightarrow \exists ag \in Agent : astate[ag] \in \{"readyAbort", "aborted"\})$
   (phrase ambiguë : la forme inverse $\exists ag \ldots \Rightarrow cstate = "aborted"$ *est acceptable aussi)*
   Aussi : $(\exists ag \in Agent : astate[ag] = "readyAbort") \leadsto cstate = "aborted"$
9. `Irrevocability` : la décision finale d'un agent (`committed` ou `aborted`) est irrévocable, il n'en change plus une fois prise.
   $\forall ag \in Agent, v \in \{"aborted", "committed"\} : \Box(astate[ag] = v \Rightarrow \Box(astate[ag] = v))$
10. `DecisionReached` : tout agent finit par décider `committed` ou `aborted`.
    $\forall ag \in Agent : \Diamond(astate[ag] = "aborted" \lor astate[ag] = "committed")$
11. `MessagesAllReceived` : tous les messages sont finalement consommés.
    $\Box\Diamond(cmsgs = EmptyBag \land \forall ag \in Agent : amsgs[ag] = \{\})$
    (note that, as channels are initially empty, a simple $\Diamond(...)$ *is trivially valid)*
12. `CanCommit` : il est possible que tous les agents valident. (Note : c'est une propriété CTL).
    $\exists\Diamond(\forall ag \in Agent : astate[ag] = "committed")$

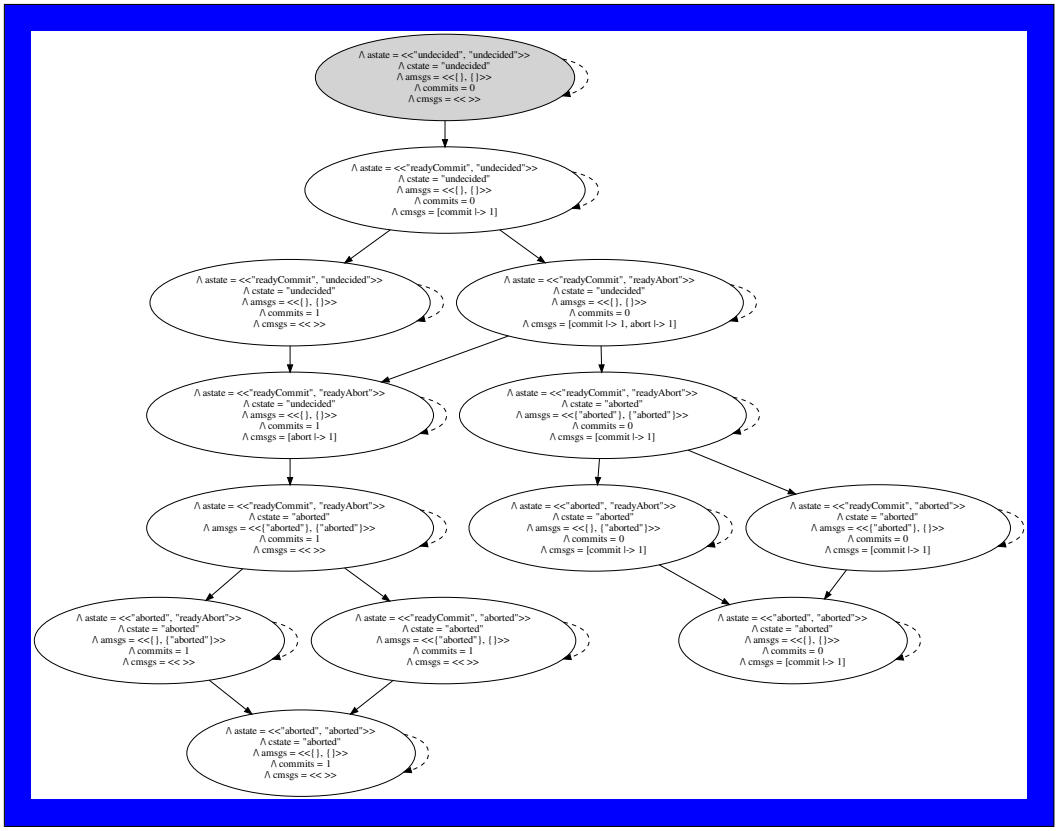
3.3 Équité

13. Énoncer l'équité minimale nécessaire pour que la propriété `DecisionReached` soit vérifiée.
    $WF_{vars}(Next)$ *(oui, c'est valide : Next est une action !)*
    ou en développant Next, équité faible sur chacune des actions.
    L'équité faible est nécessaire sur toutes les actions, car il faut que les agents proposent une valeur, que le coordinateur les récupère, qu'il réponde et qu'enfin les agents récupèrent la décision. L'équité faible est suffisante car toutes les actions sont stables : quand elles sont faisables, elles le restent jusqu'à ce qu'elles soient faites.

14. La propriété `MessagesAllReceived` est-elle vérifiée avec cette équité ?
    It's a not verified : after the coordinator gets one abort, the other messages are not read.

3.4 Vérification

15. Dessiner le graphe de transitions pour le cas particulier suivant : $N = 2$, l'agent 1 propose commit *puis* l'agent 2 propose abort (13 états).



16. Comment vérifie-t-on la propriété `FinalAgreement` ?
    Propriété de type $\Box P$ où P est un prédicat d'état. Donc regarder tous les états accessibles pour voir s'ils vérifient P.
17. Comment vérifie-t-on la propriété `DecisionReached` ?
    Propriété de vivacité de type $\Diamond P$ où P est un prédicat d'état. Donc vérifier que toute boucle (que ce soit par bégaiement ou par cycle) qui ne passe jamais par un état vérifiant P est impossible compte tenu de l'équité énoncée.
18. Comment vérifie-t-on la propriété CTL `CanCommit` ?
    On cherche un chemin qui part de l'état initial et conduit à un état vérifiant ($\forall ag \in Agent$: $astate[ag] =$ "committed").

3.5 Défaillances

19. Ajouter une action pour perdre arbitrairement un message envoyé au coordinateur.

```
MessageLoss ==
   ∃ msg ∈ BagToSet(cmsgs) :
      ∧ cmsgs' = cmsgs (-) SetToBag({msg})
      ∧ UNCHANGED <<astate, cstate, amsgs, commits>>
```

20. Parmi les propriétés `FinalAgreement`, `Irrevocability`, `DecisionReached`, `CanCommit`, lesquelles sont toujours vérifiées par le modèle ?
Plus vérifiées : `DecisionReached` (pas tous les messages).
Conservées : `FinalAgreement`, `Irrevocability`, `CanCommit` (existential property)

3.6 Module fourni : `TwoPhaseCommit.tla`

MODULE *TwoPhaseCommit*

EXTENDS *FiniteSets*, *Bags*, *Naturals*

CONSTANT N number of agents
$Agent \triangleq 1 \,..\, N$

messages used
$propose \triangleq \{\text{“commit”}, \text{“abort”}\}$
$order \triangleq \{\text{“committed”}, \text{“aborted”}\}$

VARIABLES
amsgs, messages for agent i
cmsgs, messages for the coordinator
astate, state of agent i
cstate, state of the coordinator
commits number of agents that agree to commit

$vars \triangleq \langle amsgs, cmsgs, astate, cstate, commits \rangle$

$TypeInvariant \triangleq$
$\land\ amsgs \in [Agent \to \text{SUBSET } order]$
$\land\ IsABag(cmsgs)$
$\land\ BagToSet(cmsgs) \in \text{SUBSET } propose$
$\land\ commits \in Nat$
$\land\ cstate \in \{\text{“undecided”}, \text{“aborted”}, \text{“committed”}\}$
$\land\ astate \in [Agent \to \{\text{“undecided”}, \text{“readyAbort”}, \text{“readyCommit”}, \text{“aborted”}, \text{“committed”}\}]$

$Init \triangleq \land\ amsgs = [n \in Agent \mapsto \{\}]$
$\land\ cmsgs = EmptyBag$
$\land\ astate = [n \in Agent \mapsto \text{“undecided”}]$
$\land\ cstate = \text{“undecided”}$
$\land\ commits = 0$

$AgentProposeCommit(i) \triangleq$
$\land\ astate[i] = \text{“undecided”}$
$\land\ astate' = [astate \text{ EXCEPT } ![i] = \text{“readyCommit”}]$

$\land\ cmsgs' = cmsgs \oplus SetToBag(\{\text{“commit”}\})$
$\land$ UNCHANGED $\langle cstate,\ amsgs,\ commits\rangle$

$AgentProposeAbort(i)\ \triangleq$
$\land\ astate[i] = \text{“undecided”}$
$\land\ astate' = [astate$ EXCEPT $![i] = \text{“readyAbort”}]$
$\land\ cmsgs' = cmsgs \oplus SetToBag(\{\text{“abort”}\})$
$\land$ UNCHANGED $\langle cstate,\ amsgs,\ commits\rangle$

$AgentReceiveDecision(i)\ \triangleq$
$\land\ \exists\, msg \in amsgs[i]:$
$\quad\land\ astate' = [astate$ EXCEPT $![i] = msg]$
$\quad\land\ amsgs' = [amsgs$ EXCEPT $![i] = amsgs[i] \setminus \{msg\}]$
$\quad\land$ UNCHANGED $\langle cstate,\ cmsgs,\ commits\rangle$

$CoordReceiveCommit\ \triangleq$
$\land\ cstate\ = \text{“undecided”}$
$\land\ \exists\, msg\ \in BagToSet(cmsgs):$
$\quad\land\ msg = \text{“commit”}$
$\quad\land\ commits' = commits + 1$
$\quad\land\ cmsgs' = cmsgs \ominus SetToBag(\{msg\})$
$\quad\land$ UNCHANGED $\langle astate,\ cstate,\ amsgs\rangle$

$CoordReceiveAbort\ \triangleq$
$\land\ cstate\ = \text{“undecided”}$
$\land\ \exists\, msg\ \in BagToSet(cmsgs):$
$\quad\land\ msg = \text{“abort”}$
$\quad\land\ cmsgs' = cmsgs \ominus SetToBag(\{msg\})$
$\quad\land\ cstate' = \text{“aborted”}$
$\quad\land\ amsgs' = [i \in Agent \mapsto amsgs[i] \cup \{cstate'\}]$
$\quad\land$ UNCHANGED $\langle astate,\ commits\rangle$

$CoordDecideCommit\ \triangleq$
$\land\ cstate = \text{“undecided”}$
$\land\ \ldots$

$NextAgent(i)\ \triangleq$ TRUE
$NextCoord\ \triangleq$ TRUE
$Next\ \triangleq\ \ldots$
$Spec\ \triangleq\ \ldots$